1 ***(5 puntos)*** *Considere un computador cuyo procesador, de 32 bits y reloj de 1 GHz, está basado en un pipeline de 5 etapas. Además, dispone de un sistema de memoria virtual paginada, con páginas de 16 KB, direcciones de 50 bits y tres niveles de tablas de páginas, TLBs y memorias caché separadas para instrucciones y datos. Algunas características de estos elementos que debe considerar son las siguientes:*

- *Tiempos de acceso: TLBi y TLBd 0,5 ns; Mca1I y Mca1D 1 ns; Mca2 10 ns; Mp 50 ns.*
- *El acceso a caché y la traducción en TLB pueden solaparse.*
- *Organización de las memorias caché:*
  - *Mca1I: 16 KB, bloques de 32 bytes, lectura fuera de orden (OOF) y organización directa.*
  - *Mca1D: 64 KB, bloques de 32 bytes, lectura OOF, escritura aplazada con actualización (CBWA) y organización asociativa por conjuntos de 4 bloques.*
  - *Mca2 (caché unificada de segundo nivel): 1.024 KB, bloques de 32 bytes, lectura OOF, escritura CBWA y organización asociativa por conjuntos de 4 bloques.*

a) Describa cómo se interpretan las direcciones virtuales, identificando los campos que las componen y sus tamaños. Indique también cómo interpretan las direcciones físicas las memorias Mca1I y Mca1D.

b) Calcule el menor y el mayor tiempo de acceso a memoria para la lectura de un dato en este sistema suponiendo que no se producen fallos de página. Suponga para ello que la transferencia de un bloque entre Mca2 y el primer nivel requiere 20 ns y en el intercambio entre Mp y Mca2 se invierten 100 ns.

c) Dado el fragmento de programa que se muestra a continuación, determine las dependencias de datos y de control que causan ciclos de parada en el pipeline sabiendo que no dispone de ningún mecanismo de adelantamiento y que las dependencias de control se resuelven insertando ciclos de parada. Indique cuántos ciclos de parada causa cada una de las dependencias, teniendo en cuenta que el acceso al banco de registros está dividido en semiciclos. Las etapas del pipeline son las siguientes:

- *E1: Fetch e incremento del PC.*
- *E2: Decodificación, lectura de registros y cálculo de la dirección de las instrucciones de salto.*
- *E3: Ejecución, evaluación de la condición de salto y cálculo de la dirección para las instrucciones* `ld` *y* `st`*.*
- *E4: Acceso a memoria para lectura o escritura de datos.*
- *E5: Escritura en registros.*

El código que se trata de analizar contiene un bucle que se ejecuta 100 veces, comienza en la dirección 0 y ocupa un total de 16 palabras. Se incluye a continuación:

```
(1)      add  r4,  r0,  100
(2)      or   r3,  r0,  H'3000
(3)      or   r2,  r0,  H'2000
(4)      or   r1,  r0,  H'1000
(5)buc:  ld   r11, r1,  r0
(6)      add  r11, r11, r11
(7)      ld   r12, r2,  r0
(8)      add  r12, r12, r12
(9)      sub  r13, r12, r11
(10)     st   r13, r3,  r0
(11)     add  r1,  r1,  4
(12)     add  r2,  r2,  4
(13)     add  r3,  r3,  4
(14)     sub  r4,  r4,  1
(15)     cmp  r13, r4,  r0
(16)     bgt  r13, buc
```

d) Determine el tiempo de ciclo de pipeline que sería lógico definir para este procesador y calcule el CPI que se alcanzaría al ejecutar ese código suponiendo que no se produce ningún fallo de TLB ni de caché de primer nivel.

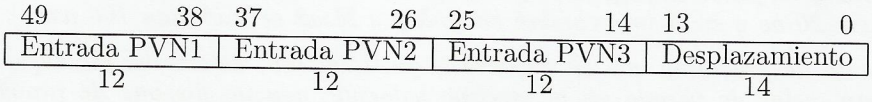
*e) Suponga ahora que se quiere determinar la ganancia (*speedup*) que se obtendría si el procesador dispusiera de mecanismos de adelantamiento y se incorporase un predictor dinámico de saltos de 1 bit inicializado a "1" (salto efectivo). Calcule dicha ganancia, así como el nuevo CPI que se obtendría al ejecutar el mismo código en este nuevo procesador (sin considerar fallos de TLB o de memoria caché).*

f) Calcule la tasa de aciertos para las memorias caché de instrucciones (Mca1I) y de datos (Mca1D) que se obtiene al ejecutar el programa anterior suponiendo que las memorias caché están inicialmente vacías. Considere que cada uno de los tres vectores que utiliza el programa ocupa 100 palabras.

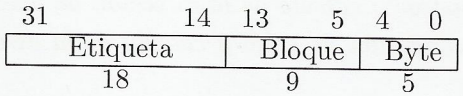
g) Suponga que el código mencionado está situado en memoria virtual ocupando el final de una página y el comienzo de la siguiente. Partiendo de que las TLB y las memorias caché están inicialmente vacías y considerando los fallos de caché obtenidos en el apartado anterior, calcule el CPI que se obtendría en el procesador que dispone de adelantamientos y predicción de salto teniendo ahora en cuenta los fallos de TLB y los de memoria caché.

SOLUCIÓN

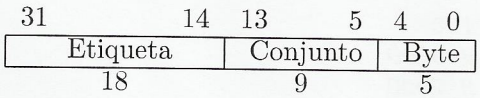
a) Al ser las direcciones virtuales de 50 bits y las páginas de 16 KB ($2^{14} Bytes$), se utilizarán 14 bits para definir el desplazamiento dentro de la página y $50 - 14 = 36$ bits para determinar la entrada de cada nivel de tablas de páginas. Suponiendo que todas las tablas de páginas de cualquier nivel sean del mismo tamaño, se reservarán $36/3 = 12$ bits para identificar la entrada de dicha tabla. En resumen, las direcciones virtuales se interpretan como sigue:



La Mca1I es directa de 16 KB ($2^{14} Bytes$) y tiene bloques de 32 B ($2^5 Bytes$), por lo que interpretará las direcciones físicas de memoria formadas por la etiqueta, el identificador del bloque y el byte dentro del bloque. Sabiendo que hay $2^{14}/2^5 = 2^9$ bloques, la interpretación es la siguiente:



La Mca1D es asociativa por conjuntos de 4, su tamaño de 64 KB ($2^{16} Bytes$) y tiene bloques de 32 B, por lo que está organizada en $2^{16}/2^5 = 2^{10}$ bloques, es decir, $2^{11}/4 = 2^9$ conjuntos. Interpretará las direcciones físicas del siguiente modo:



b) Se calcularán los tiempos mínimo y máximo de traducción y de acceso a la información.

El tiempo mínimo de traducción se produce cuando se acierta en TLB, para lo que se invierten 0,5 ns. Por otra parte, el acceso a cualquiera de las memorias caché de primer nivel se realiza en 1 ns. El enunciado indica que se puede realizar simultaneamente la traducción con el acceso a la información, por lo que la operación se completará en:

$$t_{mínimo\_lectura\_dato} = 1\ ns$$

El tiempo máximo de traducción se produce cuando hay un fallo en la TLB y hay que traducir en los 3 niveles de tablas de páginas, ubicados en memoria principal.

$$t_{máximo\_traducción} = 0,5\ ns + 3 \times 50\ ns = 150,5\ ns$$

El tiempo máximo de acceso en la lectura de un dato se produce si dicha lectura provoca fallo en Mca1D y en Mca2, y además en ambas memorias caché el bloque que se tiene que desalojar para dar entrada al nuevo bloque se encuentra modificado. En este caso, se empleará el tiempo de acceso a la Mca1D (1 ns), seguido de la escritura del bloque modificado sobre la Mca2 (20 ns), más el acceso a Mca2 para buscar el nuevo dato (10 ns) seguido de la escritura en memoria principal del bloque de Mca2 modificado (100 ns) y, finalmente, la lectura en memoria principal del dato que se buscaba (50 ns), sabiendo que en esta lectura se utiliza *"out of order fetch"*.

$$t_{máximo\_información} = 1\ ns + 20\ ns + 10\ ns + 100\ ns + 50\ ns = 181\ ns$$

Así, en total se obtiene:

$$t_{máximo\_lectura\_dato} = t_{máximo\_traducción} + t_{máximo\_información} = 150,5\ ns + 181\ ns = 331,5\ ns$$

c) Al no haber adelantamientos y estar dividido en semiciclos el acceso al banco de registros, cada dependencia de datos provocará dos ciclos de espera. Las siguientes dependencias de datos causan por lo tanto dos ciclos de parada:

- Entre las instrucciones (4) y (5) por r1.
- Entre las instrucciones (5) y (6) por r11.
- Entre las instrucciones (7) y (8) por r12.
- Entre las instrucciones (8) y (9) por r12.
- Entre las instrucciones (9) y (10) por r13.
- Entre las instrucciones (14) y (15) por r4.
- Entre las instrucciones (15) y (16) por r13.

Las dependencias de control corresponden a los saltos, por lo que únicamente se da un caso, en la instrucción (16). Puesto que se introducen ciclos de parada para resolver las dependencias de control y hasta el final de la etapa E3 no se habrá evaluado la condición de salto, se deben introducir dos ciclos de parada.

A continuación se muestran los ciclos invertidos, especificando junto al código los debidos a paradas por dependencia de datos (dd) o por dependencia de control (dc):

```
                                        Ciclos
(1)       add  r4,  r0,  100              1
(2)       or   r3,  r0,  H'3000           1
(3)       or   r2,  r0,  H'2000           1
(4)       or   r1,  r0,  H'1000           1
(5) buc:  ld   r11, r1,  r0               1+2dd (r1)
(6)       add  r11, r11, r11              1+2dd (r11)
(7)       ld   r12, r2,  r0               1
(8)       add  r12, r12, r12              1+2dd (r12)
(9)       sub  r13, r12, r11              1+2dd (r12)
(10)      st   r13, r3,  r0               1+2dd (r12)
(11)      add  r1,  r1,  4                1
(12)      add  r2,  r2,  4                1
(13)      add  r3,  r3,  4                1
(14)      sub  r4,  r4,  1                1
(15)      cmp  r13, r4,  r0               1+2dd (r4)
(16)      bgt  r13, buc                   1+2dd+2dc (r13)
```

d) A partir de los datos suministrados, el tiempo de ciclo del pipeline debería ser de 1 ns, equivalente al tiempo de acceso de las memorias caché de primer nivel y coincidente con el tiempo de ciclo del reloj del procesador. Un periodo más corto impediría acceder a memoria (para el fetch y para la etapa de acceso a datos) en un ciclo. Un periodo más largo solo tendría sentido si alguna de las etapas del pipeline tuviera una duración superior a 1 ns, pero el enunciado no especifica nada al respecto.

Si no se producen fallos de TLB ni de memorias caché de primer nivel, los ciclos de espera que se perderían al ejecutar el código proporcionado serían únicamente los mostrados en el apartado anterior, por lo que:

Instrucciones ejecutadas: $4 + 100 \times 12 = 1204$. Ciclos consumidos: $4 + 2 + 100 \times (12 + 12 + 2) = 2606$

$$CPI = 2606/1204 = 2,16$$

e) Al considerar que el procesador dispone de todos los mecanismos de adelantamiento habituales, se resuelven los ciclos de parada por dependencia de datos excepto en las dependencias entre las instrucciones (5) y (6) y entre (7) y (8), ya que en estos casos se requiere un resultado en la etapa E3 (ejecución), que se genera al final de la etapa E4 (acceso a memoria). En estos casos se introduce un único ciclo de parada. En cuanto a la única dependencia de control presente en el código, dado que ahora se dispone de un predictor dinámico de 1 bit inicializado a "salto efectivo" y realmente el salto es efectivo salvo en la última iteración del bucle, solo se producirá un fallo de predicción, es decir un total de dos ciclos de parada al ejecutar todo el código.

El nuevo CPI será:

Instrucciones ejecutadas: $4 + 100 \times 12 = 1204$. Ciclos consumidos: $4 + 100 \times (12 + 2) + 2 = 1406$

$$CPI = 1406/1204 = 1,17$$

Por consiguiente, la ganancia o *speedup* será:

$$G = 2606/1406 = 1,85$$

f) En el caso de Mca1I se realiza un total de $4 + 100 \times 12 = 1204$ accesos. Por otro lado, el número de fallos en el acceso a esta memoria caché coincidirá con el número de bloques que ocupa el código, ya que inicialmente está vacía y no se producen reemplazos de bloques de código. Hay 16 instrucciones, de una palabra cada una, comenzando al principio de un bloque (dirección 0), por lo que se ocupan $16/8 = 2$ bloques. Por consiguiente:

$$Hr_{Mca1I} = (1204 - 2)/1204 = 0,9983 \rightarrow 99,83\,\%$$

En el caso de Mca1D el número de accesos a los tres vectores durante las 100 iteraciones es $3 \times 100 = 300$. Cada vector ocupa 100 palabras, es decir $100/8 = 12,5 \rightarrow 13$ bloques, por lo que en total, los vectores ocupan $3 \times 13 = 39$ bloques. No se producen fallos por conflicto ya que la Mca1D es asociativa por conjuntos de 4 y solo se tienen 3 vectores. Tampoco hay fallos por capacidad, ya que dispone de 2^9 conjuntos y solo se ocupan 39 bloques. En resumen, se produce un total de 39 fallos, todos ellos forzosos:

$$Hr_{Mca1D} = (300 - 39)/300 = 0,87 \rightarrow 87\,\%$$

g) El número de fallos de Mca1I y de Mca1D se ha calculado en el apartado anterior: 2 y 39 respectivamente. El número de fallos de TLB depende del número de páginas de memoria virtual al que se haga referencia. Considerando el código, este hace referencia ahora a dos páginas distintas, ya que el enunciado indica que está situado al final de una y al principio de la siguiente. Por otra parte, los vectores ocupan 100 palabras y están situados en las direcciones H'1000, H'2000 y H'3000, por lo que todos ellos se encuentran en la página virtual 0 (que ocupa desde la dirección H'0000 hasta la H'3FFF) y solo provocarán un fallo de TLB de datos. En resumen, se producen:

- 2 fallos de TLBi
- 1 fallo de TLBd
- 2 fallos de Mca1I
- 39 fallos de Mca1D

El tiempo que se añade en cada fallo es el siguiente (obsérvese que todas las memorias son *OOF*:

- fallos de TLB (TLBi ó TLBd): 3×50 ns. $\rightarrow$ 150 *ciclos*
- fallos de Mca1I: $10 + 50$ ns. $\rightarrow$ 60 *ciclos*
- fallos de Mca1D: $10 + 50$ ns. $\rightarrow$ 60 *ciclos*

Por lo tanto, el nuevo CPI se calculará como en el apartado e) pero sumando ahora los ciclos perdidos en fallos de TLB y de memorias caché al total de ciclos empleados:

Instrucciones ejecutadas: $4 + 100 \times 12 = 1204$

Ciclos consumidos: $1406 + (2 + 1) \times 150 + (2 + 39) \times 60 = 1406 + 2910 = 4316$

$$CPI = 4316/1204 = 3,58$$

2 *(**3 puntos**) Sea un computador de 32 bits con un procesador capaz de ejecutar 500 MIPS y cuya Secuencia de Reconocimiento de Interrupción (SRI) dura 20 ns. Este computador se quiere utilizar como* router *de Ethernet y se le quieren conectar varios controladores de Ethernet con las siguientes características:*

- *Bloques de tamaño variable entre 64 bytes y 1.536 bytes.*
- *Velocidad de transferencia:* 10^7 *bits/s.*
- *Los registros de datos de los módulos de entrada/salida tienen un tamaño de 32 bits.*

a) *Calcule cuántas instrucciones puede ejecutar su Rutina de Tratamiento de Interrupción (RTI) si el tiempo máximo de respuesta a una solicitud de interrupción del computador es 1,5 μs.*

b) *Calcule cuántos controladores de Ethernet pueden operar en paralelo si operan mediante interrupciones sabiendo que sus Rutinas de Tratamiento de Interrupción (RTI) ejecutan 100 instrucciones.*

c) *Justifique qué tamaño deberían tener los registros de datos de los módulos de entrada/salida para que pudieran operar, al menos, 16 unidades en paralelo.*

d) *Calcule el número máximo de operaciones de entrada/salida por segundo que pueden realizar los 16 controladores Ethernet con el nuevo tamaño de los registros de datos. Para ello suponga:*

- *Tamaño de bloque 64 bytes.*
- *La rutina de programación de una operación de entrada/salida ejecuta 200 instrucciones.*
- *La rutina de finalización de una operación de entrada/salida ejecuta 100 instrucciones.*

Con el fin de registrar las operaciones de Ethernet, se conectan varias unidades de disco duro que operan por DMA, usando la técnica de robo de ciclo aislado, con las siguientes características:

- *Tiempo medio de acceso: 1 ms.*
- *Velocidad de transferencia:* $10 \cdot 10^6$ *bytes/s.*
- *Sectores de 1.024 bytes.*
- *4 registros de datos de 32 bits.*
- *La rutina de programación de una operación de entrada/salida ejecuta 200 instrucciones.*
- *La rutina de finalización de una operación de entrada/salida ejecuta 100 instrucciones.*
- *El protocolo de concesión y liberación de los buses dura 6 ns.*
- *Cada ciclo de acceso a memoria tiene una duración de 4 ns.*

e) *Calcule el número máximo de operaciones de entrada/salida por segundo que puede realizar cada unidad de disco duro.*

f) *Calcule el número máximo de discos que podrían operar simultáneamente con los 16 controladores Ethernet.*

SOLUCIÓN

a) Se calcula el tiempo entre peticiones de interrupción partiendo de la velocidad de transferencia y de la capacidad de los registros de datos:

$$Tiempo\ entre\ interrupciones = \frac{Bits\ por\ petición}{v_{transf}} = \frac{32\ bits/petición}{10^7\ bits/s} = 3,2 \cdot 10^{-6}\ s = 3,2\mu s = 3.200ns$$

A este tiempo hay que restarle el tiempo de respuesta y el tiempo empleado por la SRI. Con ese tiempo y teniendo en cuenta la capacidad de proceso del computador, se determina el número máximo de instrucciones de la RTI:

$$Número\ de\ instrucciones = (3.200ns - 1.500ns - 20ns) \times (500 \cdot 10^6 IPS \cdot 10^9 ns/s) = 1.680ns \times 0,5I/ns = 840I$$

b) Se calcula la capacidad de proceso que requiere el tratamiento de las interrupciones de un controlador Ethernet a partir de su frecuencia de petición:

$$Frecuencia\ de\ petición = \frac{v_{transf}}{Bits\ por\ petición} = \frac{10^7\ bits/s}{32\ bits/petición} = 312.500\ peticiones/s$$

Tratar cada interrupción supone realizar la SRI y ejecutar la RTI. Es decir, 20 ns y 100 instrucciones lo que equivale a 110 instrucciones en un procesador de 500 MIPS.

$$Instrucciones\ por\ segundo = 312.500\ peticiones/s \times 110\ instrucciones/petición = 34,375 \cdot 10^6\ instrucciones/s$$

Se necesita una capacidad de cómputo de 34,375 MIPS para atender las interrupciones de un controlador Ethernet. Por lo tanto, puesto que 500 MIPS/34,375 MIPS = 14,55, podrán operar en paralelo un máximo de 14 unidades con este procesador.

c) Para que puedan operar 16 controladores cada uno debería consumir como mucho 31,25 MIPS, es decir, un 90 % de lo actual. Añadiendo 1 registro de datos al módulo de E/S, se reduce a la mitad el número y la frecuencia de interrupciones, reduciéndose así el consumo de cpu a la mitad (17,1875 MIPS) y permitiendo el funcionamiento simultáneo de 29 unidades.

d) Se calcula el tiempo de una operación de Ethernet para el mínimo tamaño de bloque:

$$T_{Eth} = T_{prog} + T_{tran} + T_{última-RTI} = \frac{200\ I}{500 \cdot 10^6 IPS} + \frac{64\ B \cdot 8\ b/B}{10^7\ b/s} + (20ns + \frac{100I + 100I}{500 \cdot 10^6 IPS}) =$$

$$= 400\ ns + 51.200ns + 420ns = 52.020ns$$

De modo que un controlador Ethernet podrá realizar:

$$Operaciones\ por\ segundo = \frac{10^9\ ns/s}{52.020\ ns} = 19.223,4\ operaciones/s$$

Y los 16 simultáneamente:

$$Operaciones\ por\ segundo = 19.223,4\ operaciones/s \times 16 = 307.574,4\ operaciones/s$$

e) Se calcula el tiempo de una operación de escritura de una unidad de disco:

$$T_{UD} = T_{prog} + T_{acc} + T_{tran} + T_{RTI} = \frac{200\ I}{500 \cdot 10^6 IPS} + 1ms + \frac{1.024\ B}{10 \cdot 10^6\ B/s} + (20ns + \frac{100I}{500 \cdot 10^6 IPS}) =$$

$$400\ ns + 1ms + 102,4\mu s + 220ns = 1.103,02\mu s$$

De modo que una unidad de disco podrá realizar:

$$Operaciones\ por\ segundo = \frac{10^6\ \mu s/s}{1.103,02\ \mu s} = 906,6\ operaciones/s$$

f) La capacidad de proceso que requieren los 16 controladores de Ethernet es de $16 \times 17,1875 = 275$ MIPS, según se calculó en el apartado c)

Por lo tanto, quedan 500 - 275 = 225 MIPS para que operen los discos duros mediante DMA. La atención de las peticiones de interrupción que, en este caso, notifican el fin de las operaciones pueden ser postergadas pero no los robos de ciclos. Se calcula, cuánto tiempo se detiene al procesador por cada robo de ciclo:

$$Tiempo_{DMA} = 6\ ns + \frac{32\ bits}{32\ bits/ciclo} \times 4\ ns = 10\ ns$$

Es decir en cada robo de ciclo se transmiten 4 bytes y se detiene al procesador durante 5 instrucciones. La frecuencia de petición de robos de ciclo es $10 \cdot 10^6$B/s / 4B/robo = $2,5 \cdot 10^6$ robos/s y cada disco duro resta al procesador $2'5 \cdot 10^6$robos/s×5I/robo = 12,5 MIPS y por lo tanto podrían operar 225 MIPS/12,5 MIPS = 18 discos duros.

3 *(2 puntos) Para la implementación de una primitiva de LOCK(cerrojo) en un MP UMA con cachés privadas y un mecanismo de coherencia de cachés por actualización se consideran las tres opciones siguientes, A, B y C:*

Implementación A

```
lock:
    while (LOAD(cerrojo) == 1) do
            nothing;
    STORE (1, cerrojo); almacena un '1' en la dirección del cerrojo
```

Implementación B

```
lock:
    while (TEST&SET (cerrojo) == 1) do
            nothing;
```

Implementación C

```
lock:
    repeat
            while (LOAD(cerrojo) == 1) do
                    nothing;
    until (TEST&SET(cerrojo) == 0)
```

a) Explique brevemente el funcionamiento de las tres implementaciones. ¿Son correctas todas las implementaciones de la primitiva LOCK para el sistema considerado? Indique razonadamente por qué.

b) Para cada una de las implementaciones que considere que funcionan correctamente, explique en qué medida es eficiente desde un punto de vista del rendimiento y por qué.

SOLUCIÓN

a) Si se analizan en un primer vistazo las tres soluciones, se puede observar que en todos los casos tienen una estructura de *spin-lock*. En **A** se consulta con `load` el valor del cerrojo hasta que sea 0 y entonces se escribe 1 en la dirección correspondiente con `store`. El mecanismo de coherencia haría que esta escritura se hiciese visible en el resto de las cachés que tuviesen copia del bloque del cerrojo. Sin embargo, desde la lectura hasta la escritura no existe la garantía de que se realice sin que pueda mediar la escritura de otro procesador y, por tanto, se trata de una implementación incorrecta. En **B** el `test&set` devuelve el valor previo de la dirección y escribe indiscriminadamente un uno en ella, y con la garantía de que estas dos acciones se realizan de manera atómica, a diferencia del caso anterior y, en consecuencia, sí se trata de una realización correcta. Por último, en **C** se realiza primero la comprobación de que el cerrojo está abierto y sólo cuando así sea se ejecuta un `test&set` para tratar de ganar su acceso, en un comportamiento que es entonces análogo al caso **B** (aunque sólo para los procesadores que hubieran pasado la comprobación primera) y por tanto funcionalmente correcto.

b) Sólo consideraremos aquí las implementaciones correctas, o sea, B y C. Con *actualización* como política de coherencia de cachés todas las copias de un mismo bloque se mantienen siempre actualizadas. En el caso de **B**, todos los procesadores detenidos en el lock darán lugar a que se produzca tráfico de actualización en el sistema debido a las escrituras que se propagan. Nótese, sin embargo, que no se producirían fallos de caché adicionales a los que ocurrirían en un comportamiento "monoprocesador": si un bloque se encuentra en caché será siempre copia válida. En **C** los procesadores en el lock no producirán fallos de caché en la primera fase (*spin* en la lectura del cerrojo) y, por tanto, ahora se reduce significativamente el tráfico extra debido a la ausencia de actualizaciones.

También es interesante destacar aquí que el patrón de *sharing* o "compartición" que se presupone en el acceso a un cerrojo es habitualmente de "grano fino", independientemente de que se haya reducido el número de escrituras. En el caso de una potencial "compartición secuencial", la actualización no parece la política de coherencia adecuada, por el exceso de tráfico que añade innecesariamente en el sistema de interconexión.